The suffix-free-prefix-free hash function construction and its indifferentiability security analysis

Bagheri, Nasour; Gauravaram, Praveen; Knudsen, Lars R.; Zenner, Erik

Published in:
International Journal of Information Security

Link to article, DOI:
10.1007/s10207-012-0175-4

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

Int. J. Inf. Secur. (2012) 11:419–434
DOI 10.1007/s10207-012-0175-4

REGULAR CONTRIBUTION

The suffix-free-prefix-free hash function construction and its indifferentiability security analysis

Nasour Bagheri · Praveen Gauravaram · Lars R. Knudsen · Erik Zenner

Published online: 12 September 2012

Abstract In this paper, we observe that in the seminal work on indifferentiability analysis of iterated hash functions by Coron et al. and in subsequent works, the initial value (IV) of hash functions is *fixed*. In addition, these indifferentiability results do not depend on the *Merkle–Damgård (MD) strengthening* in the padding functionality of the hash functions. We propose a generic n-bit-iterated hash function framework based on an n-bit compression function called suffix-free-prefix-free (SFPF) that works for *arbitrary* IVs and does not possess *MD strengthening*. We formally prove that SFPF is indifferentiable from a random oracle (RO) when the compression function is viewed as a fixed input-length random oracle (FIL-RO). We show that some hash function constructions proposed in the literature fit in the SFPF framework while others that do not fit in this framework are not indifferentiable from a RO. We also show that the SFPF hash function framework with the provision of *MD strengthening* generalizes any n-bit-iterated hash function based on an n-bit compression function and with an n-bit chaining value that is proven indifferentiable from a RO.

Keywords Indifferentiability · Merkle–Damgård · MD strengthening · Random oracle · SFPF

A portion of this project and initial submission were done when the author was a postdoc researcher in the Department of Mathematics, Technical University of Denmark sponsored by Danish Council for Independent Research–Technology and Production Sciences (FTP) grant number 09-066486/FTP. Part of this work was done when the author was visiting CR RAO Advanced Institute of Mathematics, Statistics and Computer Science (AIMSCS), India. A portion of this project and initial submission were done when the author was employed in the Department of Mathematics, Technical University of Denmark.

N. Bagheri
Electrical Engineering Department, Shahid Rajaee Teacher Training University, 16788-15811 Tehran, Iran
e-mail: NBagheri@srttu.edu

P. Gauravaram
Tata Consultancy Services Innovation Labs, Tata Consultancy Services Limited, Deccan Park, Plot No. 1, Software Units Layout, Madhapur, Hyderabad 500081, India
e-mail: p.gauravaram@tcs.com

L. R. Knudsen
Department of Mathematics, Technical University of Denmark, Matematiktorvet, Building S303, 2800 Kongens Lyngby, Denmark
e-mail: lars.r.knudsen@mat.dtu.dk

E. Zenner (✉)
University of Applied Sciences Offenburg, Badstrasse 24, 77652 Offenburg, Germany
e-mail: erik.zenner@hs-offenburg.de

1 Introduction

The problem. The Merkle–Damgård (MD) hash function construction [9,19] has influenced the design of many popular hash functions such as the SHA [20] and RIPEMD [10] families. In MD hash functions, a fixed input-length compression function is iterated to hash an arbitrary length message. The MD construction has a security reduction [9,19] that shows that a collision for the hash function implies a collision for the compression function. This is achieved by including the length of the message as part of the message padding. This technique has been termed *MD strengthening* [16]. It is interesting to note that the security reduction of the MD construction is valid for arbitrary initial values (IVs). Damgård [9] also observed that a similar reduction is possible in an iterated hash function construction *if the IV is fixed* but no message length is appended. Preneel [21] recommended *fixing the IV* as well as employing *MD strengthening*, and this is also what is used for many hash functions used in practice such as the SHA and RIPEMD families.

At CRYPTO 2005, Coron et al. [7] provided a strong notion of security for hash functions, which requires a hash function to behave like a random oracle (RO) [3] when the underlying building block is a fixed input-length random oracle (FIL-RO) or an ideal cipher. The main application of this property is that any cryptographic protocol proven secure in the RO model will remain secure even if we plug in a hash function satisfying Coron et al. security notion in the place of a hash function that is assumed to be a RO. Under this notion, Coron et al. showed that MD is insecure even if the underlying compression function is an FIL-RO. They also proposed chopMD, three variants of prefix-free MD (PFMD), NMAC, and HMAC constructions as secure variants for the MD construction and proved them as ROs in the indifferentiability security framework of Maurer et al. [18]. Subsequent research either improved [4–6,11] or extended the analysis by Coron et al. to other hash function constructions [2,12].

We observe that the indifferentiability analysis of the MD variants by Coron et al. [7] and its improvements [4–6,11] *fix the IV* of the hash function constructions and do not depend on the *MD strengthening*. This observation has led to the following interesting questions: Is it important for these indifferentiability results that the IV stays fixed? If so, then a natural question may be if it is possible to find similar constructions that are indifferentiable from a RO and where it is not necessary that the IV is fixed? What are the advantages of such hash function constructions? In this paper, we aim to seek answers for these questions.

Main contribution. In this paper, we consider the scenario in which the IV of the hash function is *not fixed* in the hash function specification. We call such designs *free-IV hash functions*. In these hash functions, the IV becomes just a part of the hash function input and it is under the control of the adversaries who try to analyze the hash functions. We identify properties that are necessary and sufficient for a *free-IV* hash function to be indifferentiable from the RO. Namely, a *free-IV* iterated hash function with an underlying FIL-RO compression function must be both *prefix-free* and *suffix-free* (SFPF) to be indifferentiable from a RO. We propose a generic n-bit hash function construction called SFPF based on an n-bit FIL-RO compression function without the provision of *MD strengthening*. We formally prove that the SFPF hash function is indifferentiable from a RO when the underlying compression function is a FIL-RO. This is our main result.

Significance. The main practical benefit of the generic SFPF hash function construction is that it gives a richer space from which one can design hash functions indifferentiable from a RO. The SFPF construction generalizes both the *fixed-IV* and *free-IV* hash functions that are indifferentiable from a RO. The SFPF hash framework and its indifferentiability security allows us to better understand the effect of IVs and *MD strengthening* on the hash function constructions derived from MD in a more formal way than permitted by the prior art. This is further strengthened by the following results derived from our main result:

1. In general, the upper bound of indifferentiability of an n-bit-iterated hash function with n-bit internal state is at most $2^{n/2}$. For example, constructions such as PFMD, HMAC and NMAC have indifferentiability bound of at most $2^{n/2}$ [4–7,11]. As shown in our analysis of SFPF hash function (Sect. 4), an adversary's advantage to differentiate SFPF construction after making q queries is at most $4q^2/2^n$. This leads to an upper bound of indifferentiability of $2^{(n-2)/2}$ for the SFPF construction. In general, this is the indifferentiability bound of an $(n-2)$-bit-iterated hash function with an internal state of size $(n-2)$ bits. Thus, the indifferentiability bound of an n-bit SFPF construction is reduced by 2 bits when compared to the general indifferentiability bound of at most $2^{n/2}$ attained by most other n-bit designs with n-bit internal state. This result shows that even by exerting control over the IVs, the adversary gains negligible additional advantage to differentiate the SFPF hash function when compared to a *fixed-IV* indifferentiable hash function.
2. We show that under the SFPF framework, one variant of PFMD [7] as well as HMAC hash functions is indifferentiable when their *IV*s are set free. We demonstrate attacks that show that two variants of PFMD [7] are not indifferentiable when their *IV*s are set free. Similar attacks can be applied on chopMD and NMAC.
3. An interesting consequence of our main result is that we can show that variants of MD that are not indifferentiable from a RO in the free-IV setting (i.e., two variants of PFMD, NMAC, and chopMD [6]) as indifferentiable designs with the provision of MD strengthening. This feature bears resemblance with collision resistance security reduction of MD framework that also holds only with the provision of MD strengthening even for arbitrary IVs. Therefore, our result implicitly shows the significance of MD strengthening on the indifferentiability security of certain hash function modes.

Security proof methodology. We prove the indifferentiability security of the SFPF framework by using a game-playing argument, a method that was successfully used in the indifferentiability analysis of the MD variants [2,7,12]. Precisely, our method can be seen as a "dual" of the method used to prove the indifferentiability of the EMD construction [2]. However, the indifferentiability security proof of the SFPF hash function has some new techniques compared to those of [2,7,12] as briefly noted below:

– The game-playing methodology used in [2,7,12] assumes that the IV of these functions is *fixed* and hence is not directly useful to prove the indifferentiability of the SFPF hash function. For instance, the proofs of [2,12] use tables and a single tree structure to store adversarial queries/responses and establish connections among the table entries respectively. In contrary, we develop a series of games from scratch using multiple trees and tables to prove the indifferentiability of the SFPF hash function.
– The indifferentiability security proofs of hash functions in [2,7,12] only formally show that the so-called *message extension attack* [7,17] on the MD construction does not apply to its variants. The techniques employed in the indifferentiability analysis of the SFPF framework, however, also take into consideration other attacks such as pseudo collisions (collisions using distinct IVs) for the MD construction.

Guide to the paper. Section 2 introduces notation and definitions. In Sect. 3, the generic SFPF hash construction is introduced, and its indifferentiability security proof is provided in Sect. 4. In Sect. 5, we show the indifferentiability of some hash functions in the *free-IV* setting and constructions that do not fit into the SFPF framework. In Sect. 6, we show that the *free-IV* hash functions with *MD strengthening* are indifferentiable from a RO. Section 7 concludes the paper.

2 Preliminaries

In this section, some basic notation and definitions are introduced. Some notation specific to the indifferentiability analysis of the SFPF hash function is introduced in Sect. 4.

2.1 Notation

We denote by $X\|Y$ the concatenation of two binary bit strings X and Y, and by $X|_j$ the value of X truncated to its j lower bits. $|X|$ represents the length of the string X in bits. An empty string is denoted by Φ. Assigning to X a random value from $\{0,1\}^z$ is denoted by $X \xleftarrow{\$} \{0,1\}^z$. We denote by $X \leftarrow \{0,1\}^z$ assigning a z-bit string to X and by $X \leftarrow Y$ assigning the output of the expression Y to X. We denote by $R : \{0,1\}^* \rightarrow \{0,1\}^n$ a random oracle with n-bit output.

Any n-bit hash function mode constructed by iterating a compression function f is denoted by $H^f : \{0,1\}^* \rightarrow \{0,1\}^n$. In this paper, we use the notation H^f to represent an iterated hash function either based on a compression function f or several distinct compression functions (e.g., $f_1, f_2, \ldots$) in accordance with the context being considered. For example, when we deal with the prefix-free (resp. suffix-free) hash function, H^f refers to the prefix-free (suffix-free) hash function. Similarly, when we deal with the SFPF design, H^f refers to SFPF based on three distinct compression functions f_1, f_2 and f_3.

The input message M to a hash function is *preprocessed* as $M = M_1\|M_2\|\ldots\|M_N$ where M_i are the message blocks of m bits each and N is the total number of blocks. We assume that $|M|$ is a multiple of m, if necessary by appending a pad of type $pad = 1\|0\ldots\|0$ to M. Whenever *MD strengthening* is included as part of pad, we mention it explicitly. The maximum allowable number of message blocks in a message including padding is denoted by $N^{\max}$. The chaining values (intermediate hash values) of $H^f(M)$ are denoted by y_i for $i = 1, \ldots, N-1$ where $y_i = f(y_{i-1}, M_i)$. y_0 is the IV and y_N is the hash value.

2.2 Indifferentiability

Definition 1 [7,18] A hash function H^f with oracle access to an ideal primitive f is said to be (t_A, t_S, q, ϵ) indifferentiable from a random oracle R if there exists a simulator S, such that for any computationally unbounded distinguisher $\mathcal{A}$ with oracle access to (H^f, f) and (R, S) respectively denoted by $\mathcal{A}^{(H^f,f)}$ and $\mathcal{A}^{(R,S)}$, it holds that:

$$Adv^{indif}_{R,S}(\mathcal{A}) = \left| Pr\left[\mathcal{A}^{(H^f,f)} \Rightarrow 1\right] - Pr\left[\mathcal{A}^{(R,S)} \Rightarrow 1\right]\right| \leq \epsilon$$

The simulator has oracle access to R and runs in time at most t_S. The distinguisher $\mathcal{A}$ runs in time at most t_A and makes at most q queries. H^f is said to be (computationally) indifferentiable from R if the bound ϵ is a negligible function of the security parameter k, where in the case of hash function k is replaced by n, the output length of hash function.

We denote the maximum number of message blocks in a single query by τ where $\tau = 1$ for (f/S) and $\tau \leq N^{\max}$ for (H^f/R). In addition, we denote the total number of queried messages by q. The security parameter k refers to the size of the hash value in bits. For any function U, we denote by $\hat{U}$ the ideal U and by $\bar{U}$ either U or $\hat{U}$. For example, if H^f is a hash function, $q^{\bar{H}^f}$ denotes the total number of queries to either H^f or the ideal hash function $\hat{H}^f$. Note that $\hat{H}^f$ can be also a random oracle R.

Similar to the simulators used in the indifferentiability analysis of the MD variants [2,4,5,7,11,12], the simulator used in the indifferentiability analysis of the SFPF hash function H^f maintains a history of all previous query relations, that is, pairs of adversary queries and simulator responses. However, unlike in the prior works, the simulator presented in this paper does not know the IV value of H^f before a query has been made to H^f/R. We denote the ith query-response relation by $QR_i : \{QR_i^q \rightarrow QR_i^r\}$, where QR_i^q is the ith query and QR_i^r is the corresponding response. The simulator stores all previous distinct query-response sets QR_i in a table T, that is after each query QR_i^q, if no entry $T[QR_i^q]$ exists, a new entry is added to T.

2.3 Iterated hash function constructions

We review the MD construction and some of its indifferentiable variants proposed by Coron et al. [7]. The other schemes of Coron et al. as well as the EMD and MDP constructions are defined in "Appendix 1".

MD hashing. Let $M = M_1 \| \ldots \| M_N$ be a padded and MD strengthened message such that $|M_i| = m$ for $i = 1 \ldots N$. Given a compression function $f : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$, the MD hash value of M is computed as $\mathrm{MD}^f(M) = f(f(\ldots f(IV, M_1), \ldots), M_N)$.

PFMD constructions. Each PFMD construction uses a padding function g which ensures that for any two messages M, M' with $M \neq M'$, $g(M)$ cannot be a prefix of $g(M')$. Three variants of PFMD are described in Algorithms 5–7 in "Appendix 1".

HMAC and NMAC constructions. The HMAC hash construction (Algorithm 8 in "Appendix 1") hashes a message by applying the same MD^f function twice, using the same IV. HMAC is a special case of the NMAC construction that is not discussed in detail here.

ChopMD constructions. The ChopMD construction is an n-bit MD hash where s out of n bits of the hash value are chopped, thus producing an $(n-s)$-bit hash value. Its variant ChopPFMD does the same for the PFMD construction [6].

SFPF constructions. The SFPF construction is an n-bit-iterated hash function, with *free-IV*, which ensures that it is not feasible in polynomial time to use $H^f(IV, M)$ in the calculation of $H^f(IV', M')$ for any two messages M, M' with $M \neq M'$.

3 Building an SFPF hash function construction

In this section, we first present properties that the *free-IV* hash functions should have in order to be indifferentiable from a RO and then propose SFPF hash function construction.

Prefix-free hash functions. The variants of MD discussed in Sect. 2.3 share the unique property that their last message block is processed differently from the previous blocks in some way. For example, in $\mathrm{PFMD}^f_{g_2}(M)$, the last block of M has always a bit '1' as the prefix while all other blocks always start with a '0' bit. Hence, when an adversary $\mathcal{A}$ tries to differentiate any of these schemes from RO, the simulator S can recognize the potential last message block in $\mathcal{A}$'s queries. That is, $\mathcal{A}$ cannot use $H^f(IV, M)$ to calculate $H^f(IV, M \| M')$, for any $M' \neq \Phi$. In this sense, all the variants of MD proven indifferentiable by Coron et al. [7] are *prefix-free*. In the same way, for these MD variants, the simulator can recognize the potential *first* message block in $\mathcal{A}$'s queries, since these queries have the structure $(y_0, \hat{M})$, where y_0 is the publicly known IV and $\hat{M} \in \{0,1\}^m$ is some message block. The simulator S can then predict the probable messages that $\mathcal{A}$ can derive from its queries to S in order to compare them with the responses by R. Therefore, S can respond to $\mathcal{A}$'s queries appropriately.

Now, we formally define a more general definition of a *prefix-free hash function* compared to the PFMD construction presented in Sect. 2.3. To our knowledge, no formal definitions for prefix-free hash function and suffix-free hash function were provided in the literature although some research works [1] addressed the importance of prefix-freeness in hash functions for the security of applications.

Definition 2 A hash function H^f with oracle access to an ideal primitive f is said to be (t_A, q, ϵ) *prefix-free* hash function if given $H^f(IV, M)$, for any computationally unbounded adversary $\mathcal{A}$ with oracle access to H^f and f it holds that:

$$\begin{aligned} Adv^{PF-H^f}(\mathcal{A}) = Pr[&M \xleftarrow{\$} \{0,1\}^*; IV \xleftarrow{\$} \{0,1\}^n; \\ &(M', Y) \leftarrow \mathcal{A}(H^f(IV, M), |M|) : \\ &(M' \neq \Phi) \wedge (|(M \| M')| \\ &= O(|M|)) \wedge (H^f(IV, M \| M') = Y)] \leq \epsilon \end{aligned}$$

where, the adversary $\mathcal{A}$ runs in time at most t_A and makes at most q queries. H^f is said to be (computationally) *prefix-free* if the bound ϵ is a negligible function of the security parameter k.

Suffix-free hash functions. Any adversary $\mathcal{A}$ trying to distinguish a *free-IV* hash function (H^f, f) from (R, S) can choose any IV value in its queries. Hence, the simulator S can no longer use a known IV value to determine the start of the messages queried by $\mathcal{A}$. However, there could be other unique properties related to the first block that can be used by S to determine the potential first message block. Informally speaking, in hash functions that possess such a property, the adversary $\mathcal{A}$ cannot use $H^f(IV, M)$ to construct $H^f(IV', M')$, for any IV' and any $M' \neq \Phi$, such that $M = M'' \| M'$. We call this class of hash functions *suffix-free hash functions*.

Now, we formally define a *suffix-free hash function* as follows:

Definition 3 A hash function H^f, with oracle access to an ideal primitive f and padding function $g(.)$, is said to be (t_A, q, ϵ) *suffix-free* hash function if given $H^f(IV, M)$ and all related chaining values, for any computationally unbounded adversary $\mathcal{A}$ with oracle access to H^f and f it holds that:

$$Adv^{SF-H^f}(\mathcal{A}) = Pr[M \xleftarrow{\$} \{0,1\}^*; IV \xleftarrow{\$} \{0,1\}^n; m_1\|\ldots\|m_N \leftarrow g(m); (IV', M', Y) \leftarrow \mathcal{A}(H^f(IV, M), |M|, f(IV, m_1), f(f(IV, m_1), m_2), \ldots) : ((IV, M) \neq (IV', M')) \wedge (M' \neq \Phi) \wedge (M = M''\|M') \wedge (H^f(IV, M) = H^f(IV', M') = Y)] \leq \epsilon$$

where $g(M)$ is the padding function and the adversary $\mathcal{A}$ runs in time at most t_A and makes at most q queries. H^f is said to be (computationally) *prefix-free* if the bound ϵ is a negligible function of the security parameter k.

By assuming that the compression function is a FIL-RO, a suffix-free hash can be constructed, for example, by encoding a bit '1' and a bit '0' as the starting bit in the first and the remaining blocks of the message respectively.

Remark 1 We remark that *fixed-IV* hash functions that are prefix-free are also prefix-free when the IV is made free. However, in this setting, assuming that compression functions are FIL-ROs, *prefix-freeness* by itself may not be sufficient for the indifferentiability of these hash function modes. We show this in detail in Sect. 5 by demonstrating differentiability attacks on *free-IV* $\text{PFMD}^f_{g_1}$ and *free-IV* $\text{PFMD}^f_{g_2}$. On the other hand, *fixed-IV* hash functions that are suffix-free are not necessarily suffix-free when the IV is made free. For example, the designs $\text{PFMD}^f_{g_1}$ and *free-IV* $\text{PFMD}^f_{g_2}$ with *fixed-IV* are suffix-free but not when the IV is made free.

Below we show that an iterated hash function that is not *prefix-free* or *suffix-free* is not indifferentiable from a RO, demonstrating the significance of these properties in the design of indifferentiable iterated hash functions.

Theorem 1 *Any iterated hash function H^f that is not prefix-free or suffix-free is differentiable from RO, for any simulator.*

Proof Let H^f denote an iterated hash function based on an ideal primitive f, and R a random oracle and S a simulator for f. If H^f is not (t_A, q, ϵ) *prefix-free*, then there exists an adversary $\mathcal{A}$ running in time t_A and making at most q queries (to H^f and/or f) for which:

$$Pr[M \xleftarrow{\$} \{0,1\}^*; IV \xleftarrow{\$} \{0,1\}^n; (M', Y) \leftarrow \mathcal{A}(H^f(IV, M), |M|) : (M' \neq \Phi) \wedge (|(M\|M')| = O(|M|)) \wedge (H^f(IV, M\|M') = Y)] > \epsilon$$

Then, to differentiate (H^f, f) from (R, S), we can construct $\mathcal{A}'$ from $\mathcal{A}$, defined as follows:

1. Choose random (IV, M) and let Y be the result of applying $\bar{H}^f$ to (IV, M).
2. Use $\mathcal{A}$ to compute $M' \neq \Phi$ and $Y' = H^f(IV, M\|M')$, such that $|(M\|M')| = O(|M|)$,with probability $\epsilon' > \epsilon$.
3. queries for $\bar{H}^f(IV, M\|M')$ and receives Y''.
4. Output 1 if $Y'' = Y'$, or 0 otherwise.

In the above attack, the adversary outputs "1" if $\bar{H}^f$ is H^f with a non-negligible probability $\epsilon' > \epsilon$ whereas this probability for RO and any simulator would be 2^{-n}, because the simulator has no knowledge of M to answer adaptively. Hence, for an iterated hash function with/without a *free-IV* to be indifferentiable, it is necessary for it to be *prefix-free*. We would now like to state that $\mathcal{A}'$ is a $(t_{A'}, q', \epsilon')$ distinguisher between (H^f, f) and (R, S), for suitable $t_{\mathcal{A}'}$, q' and ϵ'.

On the other hand, if H^f is not (t_A, q, ϵ) *suffix-free*, then there exists an adversary $\mathcal{A}$ running in time t_A and making at most q queries (to H^f and/or f) for which:

$$Pr[M \xleftarrow{\$} \{0,1\}^*; IV \xleftarrow{\$} \{0,1\}^n; m_1\|\ldots\|m_N \leftarrow g(m); (IV', M', Y) \leftarrow \mathcal{A}(H^f(IV, M), |M|, f(IV, m_1), f(f(IV, m_1), m_2), \ldots) : ((IV, M) \neq (IV', M')) \wedge (M' \neq \Phi) \wedge (M = M''\|M') \wedge (H^f(IV, M) = H^f(IV', M') = Y)] > \epsilon$$

Then, to differentiate (H^f, f) from (R, S), we can construct $\mathcal{A}'$ from $\mathcal{A}$, defined as follows:

1. Choose random (IV, M) and let Y be the result of applying $\bar{H}^f$ to (IV, M).
2. Use $\mathcal{A}$ to compute $(IV', M') \neq (IV, M)$ such that $Y = H^f(IV', M')$ and $M = M''\|M'$, with probability $\epsilon' > \epsilon$.
3. queries for $\bar{H}^f(IV', M')$ and receives Y'.
4. Output 1 if $Y = Y'$, or 0 otherwise.

In the above attack, the adversary outputs "1" if $\bar{H}^f$ is H^f with a non-negligible probability $\epsilon' > \epsilon$ whereas this probability for RO and any simulator would be 2^{-n}, because finding a collision in RO is expected to cost $2^{n/2}$. Again, we would now like to state that $\mathcal{A}'$ is a $(t_{A'}, q', \epsilon')$ distinguisher between (H^f, f) and (R, S), for suitable $t_{\mathcal{A}'}$, q' and ϵ'.

Hence, for an iterated hash function with a *free-IV* to be indifferentiable, it is necessary for it to be both *prefix-free* and *suffix-free*. Otherwise, the construction would not be indifferentiable from RO, for any simulator. □

Remark 2 In the proof of Theorem 1, we have omitted the details of a padding function that may be used in a hash function that is not prefix-free or suffix-free. However, it has no influence on our analysis because the discussion easily extends to the padded version. More precisely, although different hash functions use different padding functions, many of them append a sequence of bits which includes the length of the original message to the end of the message. Hence, this sequence can also be included in the produced M'. It is exactly the approach that is used to do length extension attack on the MD hash function.

SFPF hash functions. It is possible to construct a hash function that is both suffix-free and prefix-free (SFPF) for messages of at least two blocks. For example, we can encode a message of at least two blocks in the MD construction by prefixing two 0 bits ("00") in the first block (*suffix-free padding*), prefixing a "10" pair in the last block (*prefix-free padding*) and prefixing two 1 bits ("11") in every intermediate message block. Hence, for an SFPF hash function construction, the adversary $\mathcal{A}$ cannot use $H^f(IV, M)$ to construct either $H^f(IV', M')$ (such that $M = M''\|M'$) or $H^f(IV, M\|M')$ for any IV, IV', M, and $M' \neq \Phi$.

3.1 SFPF hash function construction

The SFPF hash function can be constructed as follows: Consider a *free-IV* iterated hash function based on the FIL-RO compression function $f : \{0,1\}^{n+m} \rightarrow \{0,1\}^n$ but without *MD strengthening*. Let M be the arbitrary length message to be processed. We split M into blocks $M_1\| \ldots \|M_N$ such that each block is of size $m-2$ bits, if necessary by appending the last block M_N with *pad* bits. Recall that *pad* does not include *MD strengthening*. We let the first f process the first $m-2$ bits of M combined with the two *suffix-free padding* bits and the final f process the last message block combined with the two *prefix-free padding* bits. Each of the remaining blocks of M is prefixed with two bits, distinct from the combination of the bits used in the first and the last block, and processed by the same compression function f.

Alternatively, we can consider processing the message in the above setting with an iterated hash function based on three distinct FIL-RO compression functions $f_i : \{0,1\}^{n+m} \rightarrow \{0,1\}^n$ for $i = \{1,2,3\}$. We remark that this description is comparable with that of the 6-round Luby-Rackoff construction based on six distinct FIL-ROs [8]. In this alternative description, we divide M into m-bit blocks $M_1\| \ldots M_N$ (if necessary by appending the last block M_N with *pad* bits). We employ f_1 to process the first m-bit block, f_3 to process the last m-bit block and f_2 to process the intermediate m-bit blocks as described in Algorithm 1 and shown in Fig. 1. We denote by H^f the SFPF construction based on f_i for $i = \{1,2,3\}$, and by f we mean f_1, f_2, and f_3. Note that the PFMD$^f_{g_2}$ construction defined in Sect. 2.3 can also be seen as based on two distinct FIL-RO compression functions as it pads last block differently compared to the remaining blocks.

Algorithm 1: Hash construction SFPF

Input: $y_0 \in \{0,1\}^n$, $M = (M_1\|M_2\| \ldots \|M_N)$ where $|M_i| = m$
`// IV is free`

$y_1 \leftarrow f_1(y_0, M_1)$ `// mark the beginning`
$y_i \leftarrow f_2(y_{i-1}, M_i)$ for $2 \leq i \leq N-1$ `//` y_{i-1} `for` $2 \leq i \leq N-1$ `are the chaining values`
$y' \leftarrow f_3(y_{N-1}, M_N)$ `// mark the end`
return y'

Our SFPF hash function is generic but not the most generic SFPF. The SFPF design defined in Algorithm 1 is generic from the view that the compression functions f_1, f_2 and f_3 can be defined in different ways. Basically, for any SFPF hash function design, the way the first block and last block are processed should be distinguished in some way. As noted before, one such way is to have a single compression function in the iteration and processing the first message block with "00" as the prefix, last message block with "10" as the prefix and all intermediate blocks with "11" as the prefix. However, we remark that our SFPF based on 3 distinct compression functions is not the most generic SFPF hash function construction as all SFPF hash functions may not be of this form. That is, it may be possible to provide an SFPF scheme that does not match our framework. For example, by replacing each function f_2 in the iteration in our SFPF with a distinct compression function, an SFPF scheme with a different form can be obtained. In the following section, we consider the indifferentiability analysis of our SFPF hash function design. For any other SFPF constructions based on FIL-RO compression function, it would be possible to prove the indifferentiability following an approach similar to ours, but the details of the games should be defined based on the target SFPF construction.

4 Indifferentiability analysis of the SFPF hash function

In Theorem 2 of this section, we prove that the SFPF hash function, H^f, is indifferentiable from a random oracle R.

Theorem 2 *The SFPF hash function* $H^f : \{0,1\}^* \rightarrow \{0,1\}^n$ *based on three FIL-RO compression functions* f_1, f_2,

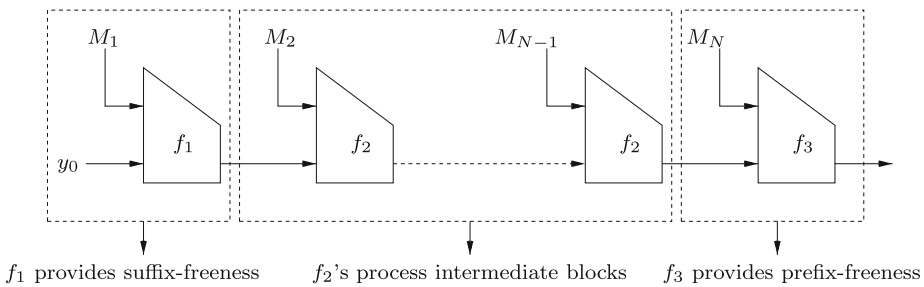

Fig. 1 The SFPF hash function construction H^f

$f_3 : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$ is (t_A, t_S, q, ϵ) indifferentiable from a random oracle R, with the same domain and range, for any t_A and $t_S \le q(q+1)/2$ with $\epsilon \le \frac{4q^2}{2^n}$ where q denotes total queried messages blocks to $\bar{H}^f$, $\bar{f}_1$, $\bar{f}_2$ and $\bar{f}_3$.

Proof In the following, we only discuss the logic of the simulator used in deriving the indifferentiability security bound for the SFPF hash function. The full proof for this Theorem has been provided in "Appendix 2".

Let S^f be a simulator which simulates f_1, f_2 and f_3. Hence, we need to program S^f such that no distinguisher $\mathcal{A}$ can distinguish (except with negligible probability) between the following two scenarios:

- $\mathcal{A}$ has oracle access to $(H^f, (f_1, f_2, f_3))$.
- $\mathcal{A}$ has oracle access to (R, S^f).

Using this experiment, we define the advantage of $\mathcal{A}$ as follows:

$$Adv^{indif}_{H^f,R}(\mathcal{A}) = \epsilon = \left| Pr[\mathcal{A}^{H^f,(f_1,f_2,f_3)} \Rightarrow 1] - Pr[\mathcal{A}^{R,S^f} \Rightarrow 1] \right|$$

The simulator does not see $\mathcal{A}$'s queries to $\bar{H}^f$ (either H^f or R); however, it can call R when needed for simulation.

Whenever $\mathcal{A}$ queries $\bar{H}^f$, it cannot directly access the output of $\bar{f}_1$ and $\bar{f}_2$. $\mathcal{A}$ must query $\bar{f}_1$ and $\bar{f}_2$ to know about their output. The simulator provides random answers to the new queries of $\mathcal{A}$ to $\bar{f}_1$ and $\bar{f}_2$. On the other hand, for $\mathcal{A}$'s queries to $\bar{f}_3$, the simulator S^f should return values in a way "consistent" with H^f and R. Hence, the only way by which $\mathcal{A}$ can fool the simulator S^f is by predicting its responses for the queries to $\bar{f}_1$ and $\bar{f}_2$ or by finding a collision in the responses of S^f (explained below). However, we formally show that neither of these events can occur with high probability. The simulator program presented in Fig. 3 shows the techniques used by S^f to respond to $\mathcal{A}$ for its queries to $\bar{f}_i$ for $i = 1, 2, 3$.

The simulator S^f keeps a history of all query/responses related to $\bar{f}_1$ in a table $T_{\bar{f}_1} : T^{Q}_{\bar{f}_1} \to T^{R}_{\bar{f}_1}$ which has three columns. The queries (y_j, M_j) are stored in the first two columns of $T_{\bar{f}_1}$, denoted $T^{Q_y}_{\bar{f}_1}$ and $T^{Q_M}_{\bar{f}_1}$. The corresponding responses are stored in the third column of $T_{\bar{f}_1}$ denoted $T^{R}_{\bar{f}_1}$. The combination of the first two columns $T^{Q_y}_{\bar{f}_1} \| T^{Q_M}_{\bar{f}_1}$, denoted by $T^{Q}_{\bar{f}_1}$, includes all points in the domain of $\bar{f}_1$ that have already been queried and $T^{R}_{\bar{f}_1}$ includes all points in the range of $\bar{f}_1$ that have been assigned as the responses of queries. We also represent the table $T_{\bar{f}_1}$ by $T_{\bar{f}_1} : T^{Q}_{\bar{f}_1} \to T^{R}_{\bar{f}_1}$. Similarly, S^f keeps a history of all query/responses related to $\bar{f}_2$ and $\bar{f}_3$ and similar notation for the respective tables can be given.

In addition, the simulator S^f maintains different tree structures that include the adversarial query-response connections. The edges of these trees represent adversarial queries, and the nodes refer to the responses of S^f. Figure 2 shows possible states of the trees obtained after several queries to $\bar{f}_1$ and $\bar{f}_2$. Since any root in the trees has started from a query to $\bar{f}_1$, any query to $\bar{f}_1$ can be considered as the root of the tree. The labels of the edges indicate that queries are answered by $\bar{f}_1$ or $\bar{f}_2$. Let M^j_l be the lth block of the j^{th} queried message and y^j_{l-1} be the corresponding internal state. Note that the superscript j refers to $''$ and $'$ in the first two several query/response connections in Fig. 2, and the superscript j is blank (for M_l, y_{l-1} and y_l) in the rightmost case of Fig. 2.

The simulator uses the functions $GetPath$ and $NewPath$ to access and update the trees respectively. We explain the functionality of $NewPath$, by considering two cases:

1. $NewPath(y^j_{l-1}, \bar{f}_1, M^j_l) \leftarrow y^j_l$
2. $NewPath(y^j_0, \bar{f}_2, M^j_r \| M^j_l) \leftarrow y^j_l$

The first case is related to a new query (y^j_{l-1}, M^j_l) to $\bar{f}_1$. If $y^j_{l-1} \in T^{Q_y}_{\bar{f}_1}$, the simulator draws a new edge from the

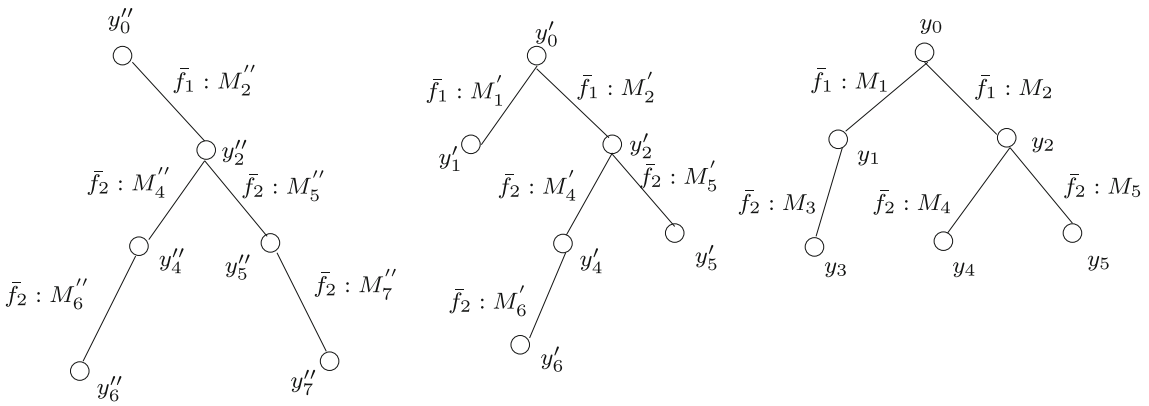

Fig. 2 A sample tree structure that can be used by a simulator in the indifferentiability analysis of SFPF

current root labeled y_{l-1}^j to a new node labeled y_l^j, and the label of the edge would be $\bar{f}_1 : M_l^j$. Otherwise, the simulator uses this new query to $\bar{f}_1$ to grow a new tree where a node labeled y_{l-1}^j indicates its root. The simulator adds an edge labeled $\bar{f}_1 : M_l^j$ from this node to a new node labeled y_l^j, which is the returned value for the current query.

The second case is related to a new query (y_{l-1}^j, M_l^j) to $\bar{f}_2$ for which there is a path from the root labeled y_0^j. The message blocks that are used in the labels of edges that are included in that path are concatenated as M_r^j. Hence, M_r^j can include more than one block of message. If such a path exists, the simulator adds an edge labeled $\bar{f}_2 : M_l^j$ from the node labeled with y_{l-1}^j to a new node labeled y_l^j.

The function $GetPath(Y)$ for some chaining value $Y \in \{0,1\}^n$ returns a sequence of message blocks that connect the chaining value Y to the root of the tree. For example, the function $GetPath(y_{l-1}^j)$ returns a sequence of message blocks, M_r^j, used as the labels in a path having edges from the root labeled y_0^j to a node labeled y_{l-1}^j. In the case of a duplicate path (two different paths that can be tracked from roots to y_{l-1}^j which requires a collision in the output of S^f) or no path (we cannot find any path from the root to y_{l-1}^j), $GetPath(Y)$ returns $Error$ and false respectively.

It must be noted that if a collision occurs in the output of S^f, then $NewPath$ may not be able to draw a new edge related to the point properly and it will file, because we have two nodes with the same label and the simulator does not know the new edge should be connected to which one. Hence, in this case, simulator fails. Therefore, the required number of queries to find a collision in the output of S^f is a trivial upper bound for indifferentiability of the scheme, this bound is $q = 2^{n/2}$. Precisely, the following bad events may disrupt the simulator's functionality:

1. On a new query $\bar{f}_1(y_{l-1}^j, M_l^j)$, if the returned value y_l^j collides with a label of a tree maintained by simulator or the input of a query to $\bar{f}_2$ or $\bar{f}_3$ which is not included in any tree. This bad event is indicated by $bad_{\bar{f}_1}$ in Fig. 3.
2. On a new query $\bar{f}_2(y_{l-1}^j, M_l^j)$, if the returned value y_l^j collides with a label of a tree maintained by simulator or the input of a query to $\bar{f}_2$ or $\bar{f}_3$ which is not included in any tree. This bad event is indicated by $bad_{\bar{f}_2}$ in Fig. 3.

In addition, finding a fixed point in the output of S^f increases the size of the tree uncontrollably but for any query to S^f it occurs with the probability of 2^{-n} which is negligible and we omit it (for the given compression function f, the pair of chaining value y_i and a message block m_i is called a fixed point when we have $f(y_i, m) = y_i$). □

Remark 3 Due to the birthday paradox, the upper bound of indifferentiability of any n-bit hash function with an n-bit chaining value is at most $2^{n/2}$ (schemes in [2,7,12] have this

On query to $\bar{f}_1(y_{l-1}^j, M_l^j)$

1. $y_l^j \xleftarrow{\$} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_1}^Q$
   - $y_l^j \leftarrow T_{\bar{f}_1}^R$
   - **ret** y_l^j
3. $NewPath(y_{l-1}^j, \bar{f}_1, M_l^j) \leftarrow y_l^j$
4. $\bar{f}_1(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
5. if $(y_l^j \in T_{\bar{f}_1}^R) \vee (y_l^j \in T_{\bar{f}_2}^{Q_y}) \vee (y_l^j \in T_{\bar{f}_2}^R) \vee (y_l^j \vee T_{\bar{f}_3}^{Q_y})$
   - $bad_{\bar{f}_1} \leftarrow true$
   - $y_l^j \xleftarrow{\$} \{0,1\}^n \backslash (T_{\bar{f}_1}^R \bigcup T_{\bar{f}_2}^{Q_y} \bigcup T_{\bar{f}_2}^R \bigcup T_{\bar{f}_3}^{Q_y})$
6. **ret** y_l^j

On query to $\bar{f}_2(y_{l-1}^j, M_l^j)$

1. $y_l^j \xleftarrow{\$} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_2}^Q$
   - $y_l^j \leftarrow T_{\bar{f}_2}^R$
   - **ret** y_l^j
3. else
   - $(y_0^j, M_r^j) \leftarrow GetPath(y_{l-1}^j)$
   - $NewPath(y_0^j, \bar{f}_2, M_r^j \| M_l^j) \leftarrow y_l^j$
4. if $y_l^j \in (T_{\bar{f}_1}^R) \vee (T_{\bar{f}_2}^R) \vee (T_{\bar{f}_2}^{Q_y}) \vee (T_{\bar{f}_3}^{Q_y})$
   - $bad_{\bar{f}_2} \leftarrow true$
   - $y_l^j \xleftarrow{\$} \{0,1\}^n \backslash (T_{\bar{f}_1}^R \bigcup T_{\bar{f}_2}^R \bigcup T_{\bar{f}_2}^{Q_y} \bigcup T_{\bar{f}_3}^{Q_y})$
5. **ret** y_l^j

On query to $\bar{f}_3(y_{l-1}^j, M_l^j)$

1. $y_l^j \xleftarrow{\$} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_3}^Q$
   - $y_l^j \leftarrow T_{\bar{f}_3}^R$
   - **ret** y_l^j
3. else if $(y_0^j, M_r^j) \leftarrow GetPath(y_{l-1}^j)$ then
   - $y_l^j \leftarrow R(y_0^j, M_r^j \| M_l^j)$
4. $\bar{f}_3(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
5. **ret** y_l^j

Fig. 3 Simulator for the SFPF hash function

bound). For the SFPF hash function, this bound is $2^{n/2}/\sqrt{4} = 2^{(n-2)/2}$, a loss of only 3 bits in security compared to the maximum achievable indifferentiability upper bound for any n-bit hash function with an n-bit chaining value. On the other hand, for the same computational work as a *fixed-IV* hash, the given SFPF scheme can hash n extra message bits via its IV at the expense of padding every block with two bits or using 3 distinct FIL-ROs. However, this is not a huge penalty as the message blocks of $\text{PFMD}^f_{g_2}$ and $\text{PFMD}^f_{g_3}$ [7] also use additional padding and counter bits respectively (for the details of $\text{PFMD}^f_{g_2}$ and $\text{PFMD}^f_{g_3}$ refer to Algorithms 7 and 6 in "Appendix 1" respectively).

Remark 4 Due to the birthday paradox, the upper bound of the security of the given SFPF hash function against generic attacks such as length extension, multicollisions [13], long message second preimages [15] and herding [14] is similar to the security of plain MD against these attacks. It must be noted that the adversary advantage after q queries to the compression function is upper bounded by $q^2/2^{n/2}$ [13] which does not compromise the claimed bound of indifferentiability of our scheme.

5 Hash function constructions (not) fitting in the SFPF framework

In this section, we reason that *free-IV* versions of the $\text{PFMD}^f_{g_3}$ and HMAC schemes fit in the SFPF framework and are hence indifferentiable. We differentiate *free-IV* $\text{PFMD}^f_{g_1}$ and $\text{PFMD}^f_{g_2}$, showing that they do not fit in the SFPF framework. Similar attacks apply to their chopped variants, NMAC and chopMD.

Indifferentiability of free-IV $\text{PFMD}^f_{g_3}$. In the $\text{PFMD}^f_{g_3}$ construction, each query to f/S is of the form (y_{j-1}, M_j, N, j), where N and j are the counter index and message length respectively. In particular, the existence of these parameters allows the simulator to identify potential first or last blocks of the message and thus the possible messages that can be built from the previous queries. From [7], we know that $\text{PFMD}^f_{g_3}$ is *prefix-free*. Since each iteration of this construction processes a counter index and the total message length, it is trivial that $\mathcal{A}$ cannot use $H^f(IV, M)$ to calculate $H^f(IV', M')$ (such that $M = M''\|M'$), for any IV' and $M' \neq \Phi$. Hence, $\text{PFMD}^f_{g_3}$ construction is also *suffix-free*.

Indifferentiability of free-IV HMAC. In the HMAC scheme (i.e., $\text{HMAC}^f(IV, M)$ for any $IV \in \{0,1\}^n$), the first and the last iteration of the compression function receive the same chaining value, denoted by IV. Hence, the simulator can use this characteristic of $\text{HMAC}^f(IV, M)$ to determine the start and the end of potential messages that can be employed to distinguish whether adversary interacts with S/f. However, the simulator can be fooled if the adversary can find a fixed point in the output of S (either a single-block fixed point or a multi-block fixed point), because, in this case, the simulator cannot determine the exact size of message which the adversary queries to R. However, since f is a *FIL-RO*, finding a fixed point costs 2^n calls to the compression function which is far beyond the indifferentiability bound for HMAC. However, if it is easy to find a fixed point in the compression function, for example, Davies Meyer (*DM*) compression function, then HMAC with *free-IV* can be easily differentiated as follows:

1. query for $(S^E/E)^{-1}(M, 0)$ for any $M \in \{0,1\}^n$ and receive X, where S^E simulates the ideal cipher E.
2. assign X to the IV value and query for $\overline{\text{HMAC}}^f(IV, M)$ and receive Y.
3. query for $\overline{\text{HMAC}}^f(IV, M\|M)$ and receive Z.
4. output "1" if $Y = Z$ else output "0".

In the case of *DM*, the adversary outputs "1" with probability 1 while for a random oracle and for any simulator this probability is 2^{-n}. Hence, the *free-IV* HMAC with *DM* as the compression function is differentiable from random oracle. However, this does not contradict our claim because *DM* is not a *FIL-RO*.

Differentiability attacks on free-IV $PFMD^f_{g_1}$ *and* $free-IV\ PFMD^f_{g_2}$. Algorithm 2 describes a distinguisher that differentiates $\text{PFMD}^f_{g_1}$ from R which implements a pseudo-collision attack against $\text{PFMD}^f_{g_1}$. The probability that an adversary $\mathcal{A}$ outputs a bit '1' is always 1 whenever it interacts with $(\text{PFMD}^f_{g_1}, f)$ and is negligible whenever $\mathcal{A}$ interacts with (R, S). Consequently, the adversary's advantage is close to 1, and $\text{PFMD}^f_{g_1}$ is differentiable from R. Similarly, Algorithm 3 describes a distinguisher for $\text{PFMD}^f_{g_2}$. The above attacks are the variants of the generic attack on the hash functions that are not *suffix-free* provided in the proof of Theorem 1.

6 Strengthening differentiable $free\text{-}IV$ hash functions

The *free-IV* hash constructions that are shown differentiable in Sect. 5 are *prefix-free*. If their structure is modified such that they are also *suffix-free*, then they fall under the SFPF hash function framework. One way of achieving this is to use *MD strengthening*. This prevents the adversary from choosing $H^f(IV, M)$ to calculate $H^f(IV', M')$ (such that $M = M''\|M'$) for any IV, IV', M, and $M' \neq \Phi$. Hence, the *free-IV* variants of these proposals with the provision of *MD strengthening* are SFPF. In Algorithm 4, we generalize these constructions by an n-bit hash function called SFPF-

Algorithm 2: Adversary against *free-IV* PFMD^f_{g1}

$y_0 \leftarrow \{0,1\}^n$, $M_0 \leftarrow 2$, $M_1 \leftarrow 1$, $M_2 \xleftarrow{\$} \{0,1\}^m$
Query (y_0, M_0) to (f/S) and obtain response y_1.
Query $(y_1, M_1 \| M_2)$ to $(\overline{\text{PFMD}}^f_{g1})$ and obtain response Y.
Query $(y_0, M_0 \| M_1 \| M_2)$ to $(\overline{\text{PFMD}}^f_{g1})$ and obtain response Y'.
if $Y = Y'$ **then return** 1 **else return** 0

Algorithm 3: Adversary against *free-IV* PFMD^f_{g2}

$y_0 \leftarrow \{0,1\}^n$
Query $(y_0, 0^m)$ to (f/S) and obtain response y_1.
Query $(y_0, 0^{m-1} \| 0^{m-1})$ to $(\overline{\text{PFMD}}^f_{g2})$ and obtain response Y.
Query $(y_1, 0^{m-1})$ to $(\overline{\text{PFMD}}^f_{g2})$ and obtain response Y'.
if $Y = Y'$ **then return** 1 **else return** 0

Algorithm 4: SFPF-N: An SFPF hash construction with the MD strengthening

Input: $y_0 \in \{0,1\}^n$, $M = (M_1 \| M_2 \| \dots \| M_N)$ where $|M_i| = m$
`// IV is free`
$y_i \leftarrow f_1(y_{i-1}, M_i)$ for $1 \le i \le N$
$y_N \leftarrow f_2(y_N, N)$ `// mark the end`
return y_N

N wherein the final compression function is denoted by f_2 and other compression functions by f_1. The indifferentiability security proof of SFPF-N is similar to that of the SFPF hash function, and the simulator functionality is disclosed in Algorithm 11 in "Appendix 3".

.

7 Conclusion

We proposed a new generic iterated hash function framework called the SFPF construction that works for arbitrary IVs without the provision of *MD strengthening* and proved that it is indifferentiable from a RO when the compression function is an FIL-RO. This result demonstrates that it is possible to design hash functions indifferentiable from a RO wherein the IV of the hash functions need not be fixed. The positive outcome of this result is that the SFPF framework with *MD strengthening* generalizes n-bit hash functions based on n-bit compression functions and with n-bit state that are proven indifferentiable from a RO.

Acknowledgments We would like to thank anonymous reviewers for their valuable comments on the paper. We also thank Colin Boyd and Choudary Gorantla for their comments on an earlier version of this paper and Shoichi Hirose for his discussions on this topic.

8 Appendix 1: Hash function constructions

Algorithms 5–10 describe some hash constructions used in the main text.

Algorithm 5: Hash construction PFMD^f_{g1}

Input: $y_0 = IV$, $M = (M_1 \| M_2 \| \dots \| M_N)$ where $|M_i| = m$
$g_1(M) \leftarrow (N \| M_1 \| M_2 \| \dots \| M_N)$ `// prepend msg length`
$y \leftarrow MD^f(y_0, g_1(M))$ `// run standard MD`
return y

Algorithm 6: Hash construction PFMD^f_{g2}

Input: $y_0 = IV$, $M = (M_1 \| M_2 \| \dots \| M_N)$ where $|M_i| = m - 1$
$g_2(M) \leftarrow ((0\|M_1)\|(0\|M_2)\| \dots \|(0\|M_{N-1})\|(1\|M_N))$
`// mark last block`
$y \leftarrow MD^f(y_0, g_2(M))$ `// run standard MD`
return y

Algorithm 7: Hash construction PFMD^f_{g3}

Input: $y_0 = IV$, $M = (M_1 \| M_2 \| \dots \| M_N)$ where $|M_i| = m$
for $i \leftarrow 1$ **to** N **do** `// for each block:`
  $y_i \leftarrow f(y_{i-1}, M_i, N, i)$ `// compress with index and msg length`
end
return y_N

Algorithm 8: Hash construction HMAC

Input: $y_0 = IV$, $M = (M_1 \| M_2 \| \dots \| M_N)$ where $|M_i| = m$
$M' = (0^m \| M_1 \| M_2 \| \dots \| M_N)$ `// prepend block of zeros`
$y \leftarrow MD^f(y_0, M')$ `// run standard MD`
if $n < m$ **then** `// ensure length(y')=m`
  $y' \leftarrow y \| 0^{m-n}$
else
  $y' \leftarrow y|_m$
$y'' \leftarrow MD^f(y_0, y')$ `// run MD`
return y''

Algorithm 9: Hash Construction EMD

Input: $y_0 = IV_0$, $y_1 = IV_1$, $M = (M_1 \| M_2 \| \dots \| M_N)$ where $|M_1| = \dots = |M_{N-1}| = m$, $|M_N| = m - n - 64$
$M'_N \leftarrow M_N \| |M|$ `// append msg length as 64-bit value`
$y \leftarrow MD^f(y_0, (M_1 \| \dots \| M_{N-1}))$ `// run standard MD`
$y' \leftarrow f(y_1, y \| M'_N)$ `// process final block with msg length`
return y'

Algorithm 10: Hash Construction MDP

Input: $y_0 = IV$, $M = (M_1 \| M_2 \| \dots \| M_N)$ where $|M_i| = m$ and M_N contains $|M|$
$y \leftarrow MD^f(y_0, (M_1 \| \dots \| M_{N-1}))$ `// run standard MD`
$y' \leftarrow f(\pi(y), M_N)$ `// permute final chaining value`
return y'

9 Appendix 2: Proof of Theorem 2

In the following, we provide a proof for Theorem 2.

Proof Let $\mathcal{A}$ be the adversary whose goal is to differentiate $(H^f, (f_1, f_2, f_3))$ from (R, S^f) by asking q non-repetitive queries where $q = \tau q^{\bar{H}^f} + q^{\bar{f}_1} + q^{\bar{f}_2} + q^{\bar{f}_3}$. Recall from the previous discussion that the simulator S^f answers each new query of $\mathcal{A}$ to $\bar{f}_1$ and $\bar{f}_2$ with a random value. The simulator S^f answers each new query to $\bar{f}_3$ by checking the possibility of the combination of the current query with the previous entries in $T_{\bar{f}_1}$, $T_{\bar{f}_2}$ and $T_{\bar{f}_3}$ in order to be in consistence with the queries to $\bar{H}^f$ and R. Since the number of entries in the tables $T_{\bar{f}_1}$, $T_{\bar{f}_2}$ and $T_{\bar{f}_3}$ together would not be more than q, the running time of the simulator is $t_S \leq q(q+1)/2$. The time of $\mathcal{A}$ to maximize its advantage for q-queries, t_A, can be any value.

We analyze the advantage of $\mathcal{A}$ by considering Games G_i for $i \in \{0, 1, \ldots, 7\}$ that are informally described in the following (formal descriptions of the games are given in "Appendix 4"). We will denote $\mathcal{A}$ with access to (playing) Game G_i by $\mathcal{A}^{G_i}$. For each game G_i, we let $p_i = Pr[\mathcal{A}^{G_i} \Rightarrow 1]$. We start with the game G_0 which directly communicates with $(H^f, (f_1, f_2, f_3))$ and complete the proof with the game G_7 which emulates (R, S^f). The intermediate games $G_1, \ldots, G_6$ would slowly transform these games into each other. We start the game playing as follows:

- **Game 0** (G_0)**:** This game shows the communication of $\mathcal{A}$ with H^f, f_1, f_2 and f_3.
- **Game 1** (G_1)**:** We denote by IH a subroutine that emulates the iteration process of the SFPF hash function H^f. This game exactly emulates H^f and f_1, f_2 and f_3. It is identical to G_0 except that FIL-ROs f_1, f_2 and f_3 are chosen in a "lazy" manner. Namely, we introduce a controller C_H that keeps the history of all queries to $\bar{f}_1$, $\bar{f}_2$ and $\bar{f}_3$ in the Tables $T_{\bar{f}_1}$, $T_{\bar{f}_2}$, and $T_{\bar{f}_3}$ respectively. Initially, the tables are empty. Upon receiving a query from $\mathcal{A}$ to $\bar{f}_1$, $\bar{f}_2$ or $\bar{f}_3$, C_H first checks in their respective tables $T_{\bar{f}_1}$, $T_{\bar{f}_2}$, or $T_{\bar{f}_3}$ for an entry corresponding to the query and if found, C_H returns that entry to $\mathcal{A}$ consistently. Otherwise C_H returns a random value for the query. In addition, C_H uses a subroutine, denoted IH, that emulates the iteration process of H^f to answer the queries of $\mathcal{A}$ to $\bar{H}^f$. Now, we can see that G_1 is a syntactic representation of G_0. Thus, $p_1 = p_0$.
- **Game 2** (G_2)**:** This game is identical to G_1 except that C_H maintains trees to detect the connection between queries and responses. The functions $GetPath$ and $NewPath$ (explained before) are used to access and update the trees respectively. The only change from G_1 to G_2 is the access and update of the trees for new queries to $\bar{f}_1$, $\bar{f}_2$ and $\bar{f}_3$. However, it has no affect on the random selection of the values returned to $\mathcal{A}$. Thus, $p_2 = p_1$.
- **Game 3** (G_3)**:** In this Game, C_H does not let IH to directly communicate with $\bar{f}_1$, $\bar{f}_2$ and $\bar{f}_3$, but it changes IH such that the FIL-ROs are simulated in IH. However, it has no effect on the returned values to $\mathcal{A}$. Thus, $p_3 = p_2$.
- **Game 4** (G_4)**:** This game is identical to G_3 except that for the new queries to $\bar{f}_3$, C_H accesses the trees to find a root connected to the current query to $\bar{f}_3$. If C_H finds such path, it concatenates the message blocks that included in that path with the current message queried to $\bar{f}_3$ and queries it to IH. However, for this query, IH returns a random value and it does not change $\mathcal{A}$'s advantage in comparison with G_3. Thus, $p_4 = p_3$.
- **Game 5** (G_5)**:** In this game, C_H applies some restriction on the values returned to $\mathcal{A}$. The controller C_H restricts the returned values for a query to $\bar{f}_1$ or $\bar{f}_2$ to not collided with any value in $T^R_{\bar{f}_1} \bigcup T^{Q_y}_{\bar{f}_2} \bigcup T^R_{\bar{f}_2} \bigcup T^{Q_y}_{\bar{f}_3}$. In general, any event that lets C_H terminate the game is considered as a *bad* event. Such events are explained below.
  It is obvious that G_4 and G_5 are identical until a bad event is set to true in G_5. This is denoted by $bad \leftarrow$ true. Hence, the maximum advantage of $\mathcal{A}$ in distinguishing G_5 from G_4 (transient from G_4 to G_5) is at most the maximum probability of the occurrence of *bad* events. Thus:

$$\left| Pr[\mathcal{A}^{G_5} \Rightarrow 1] - Pr[\mathcal{A}^{G_4} \Rightarrow 1] \right| \leq Pr[\mathcal{A}^{G_5} \Rightarrow (bad \leftarrow true)]$$

The probability that the bad events (explained below) $bad_{\bar{f}_1}$ or $bad_{\bar{f}_2}$ are set to true in G_5 is denoted as $Pr^{bad_{\bar{f}_1}}_{G_5}$ and $Pr^{bad_{\bar{f}_2}}_{G_5}$ respectively. Thus:

On query (y^j_0, M^j) to $\bar{H}^f$

1. $M^j_1 \| \ldots \| M^j_N \overset{pre}{\leftarrow} M^j$
   (a) $y^j_1 \leftarrow \bar{f}_1(y^j_0, M^j_1)$
   (b) if $N \geq 3$
      i. for $2 \leq i \leq N-1$
         A. $y^j_i \leftarrow \bar{f}_2(y^j_{i-1}, M^j_i)$
   (c) $y^j_N \leftarrow \bar{f}_3(y^j_{N-1}, M^j_N)$
2. **ret** y^j_N

On query to $\bar{f}_1(y^j_{l-1}, M^j_l)$

1. $y^j_l \leftarrow \bar{f}_1(y^j_{l-1}, M^j_l)$
2. **ret** y^j_l

On query to $\bar{f}_2(y^j_{l-1}, M^j_l)$

1. $y^j_l \leftarrow \bar{f}_2(y^j_{l-1}, M^j_l)$
2. **ret** y^j_l

On query to $\bar{f}_3(y^j_{l-1}, M^j_l)$

1. $y^j_l \leftarrow \bar{f}_3(y^j_{l-1}, M^j_l)$
2. **ret** y^j_l

Fig. 4 G_0 representation

Fig. 5 G_1 (*boxes* removed) ad G_2 (*boxes* included) representation

On query (y_0^j, M^j) to $\bar{H^f}$

1. $M_1^j \| ... \| M_N^j \overset{pre}{\leftarrow} M^j$
2. **ret** $IH(y_0^j, M_1^j \| ... \| M_N^j)$

On query to $IH(y_0^j, M_1^j \| ... \| M_N^j)$

1. $y_1^j \leftarrow \bar{f}_1(y_0^j, M_1^j)$
2. if $N \geq 3$
   (a) for $2 \leq i \leq N-1$
      i. $y_i^j \leftarrow \bar{f}_2(y_{i-1}^j, M_i^j)$
3. $y_N^j \leftarrow \bar{f}_3(y_{N-1}^j, M_N^j)$
4. **ret** y_N^j

On query to $\bar{f}_1(y_{l-1}^j, M_l^j)$

1. $y_l^j \overset{\$}{\leftarrow} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_1}^Q$
   – $y_l^j \leftarrow T_{\bar{f}_1}^R$
   – **ret** y_l^j
3. else $NewPath(y_{l-1}^j, \bar{f}_1, M_l^j) \leftarrow y_l^j$
4. $\bar{f}_1(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
5. **ret** y_l^j

On query to $\bar{f}_2(y_{l-1}^j, M_l^j)$

1. $y_l^j \overset{\$}{\leftarrow} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_2}^Q$
   – $y_l^j \leftarrow T_{\bar{f}_2}^R$
   – **ret** y_l^j
3. else if $(y_0^j, M_r^j) \leftarrow GetPath(y_{l-1}^j)$ then
   – $NewPath(y_0^j, \bar{f}_2, M_r^j \| M_l^j) \leftarrow y_l^j$
4. $\bar{f}_2(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
5. **ret** y_l^j

On query to $\bar{f}_3(y_{l-1}^j, M_l^j)$

1. $y_l^j \overset{\$}{\leftarrow} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_3}^Q$
   – $y_l^j \leftarrow T_{\bar{f}_3}^R$
   – **ret** y_l^j
3. $\bar{f}_3(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
4. **ret** y_l^j

Fig. 6 G_3 (*boxes* removed) ad G_4 (*boxes* included) representation

On query (y_0^j, M^j) to $\bar{H^f}$

1. $M_1^j \| ... \| M_N^j \overset{pre}{\leftarrow} M^j$
2. **ret** $IH(y_0^j, M_1^j \| ... \| M_N^j)$

On query to $IH(y_0^j, M_1^j \| ... \| M_N^j)$

1. $y_1^j \overset{\$}{\leftarrow} \{0,1\}^n$
2. if $(y_0^j, M_1^j) \in T_{\bar{f}_1}^Q$
   – $y_1^j \leftarrow T_{\bar{f}_1}^R$
3. $\bar{f}_1(y_0^j, M_1^j) \leftarrow y_1^j$
4. if $N \geq 3$
   (a) for $2 \leq i \leq N-1$
      i. $y_i^j \overset{\$}{\leftarrow} \{0,1\}^n$
      ii. if $(y_{i-1}^j, M_i^j) \in T_{\bar{f}_2}^Q$
         – $y_i^j \leftarrow T_{\bar{f}_2}^R$
      iii. $\bar{f}_2(y_{i-1}^j, M_i^j) \leftarrow y_i^j$
5. $y_N^j \overset{\$}{\leftarrow} \{0,1\}^n$
6. if $(y_{N-1}^j, M_N^j) \in T_{\bar{f}_3}^Q$
   – $y_N^j \leftarrow T_{\bar{f}_3}^R$
7. $\bar{f}_3(y_{N-1}^j, M_N^j) \leftarrow y_N^j$
8. **ret** y_N^j

On query to $\bar{f}_1(y_{l-1}^j, M_l^j)$

1. $y_l^j \overset{\$}{\leftarrow} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_1}^Q$
   – $y_l^j \leftarrow T_{\bar{f}_1}^R$
   – **ret** y_l^j
3. else $NewPath(y_{l-1}^j, \bar{f}_1, M_l^j) \leftarrow y_l^j$
4. $\bar{f}_1(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
5. **ret** y_l^j

On query to $\bar{f}_2(y_{l-1}^j, M_l^j)$

1. $y_l^j \overset{\$}{\leftarrow} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_2}^Q$
   – $y_l^j \leftarrow T_{\bar{f}_2}^R$)
   – **ret** y_l^j
3. else if $(y_0^j, M_r^j) \leftarrow GetPath(y_{l-1}^j)$ then
   – $NewPath(y_0^j, \bar{f}_2, M_r^j \| M_l^j) \leftarrow y_l^j$
4. $\bar{f}_2(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
5. **ret** y_l^j

On query to $\bar{f}_3(y_{l-1}^j, M_l^j)$

1. $y_l^j \overset{\$}{\leftarrow} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_3}^Q$
   – $y_l^j \leftarrow T_{\bar{f}_3}^R$
   – **ret** y_l^j
3. else if $(y_0^j, M_r^j) \leftarrow GetPath(y_{l-1}^j)$ then
   – $y_l^j \leftarrow IH(y_0^j, M_r^j \| M_l^j)$
4. $\bar{f}_3(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
5. **ret** y_l^j

Fig. 7 G_6 and G_5 representation with their IH subfunctions

On query (y_0^j, M^j) to $\bar{H}^f$

1. $M_1^j \| ... \| M_N^j \overset{pre}{\leftarrow} M^j$
2. **ret** $IH(y_0^j, M_1^j \| ... \| M_N^j)$

Only for G_5**:** On query to $IH(y_0^j, M_1^j \| ... \| M_N^j)$

1. $y_1 \overset{\$}{\leftarrow} \{0,1\}^n$
2. if $(y_0^j, M_1^j) \in T^Q_{\bar{f}_1}$
   - $y_1^j \leftarrow T^R_{\bar{f}_1}$
3. $\bar{f}_1(y_0^j, M_1^j) \leftarrow y_1^j$
4. if $N \geq 3$
   (a) for $2 \leq i \leq N-1$
      i. $y_i^j \overset{\$}{\leftarrow} \{0,1\}^n$
      ii. if $(y_{i-1}^j, M_i^j) \in T^Q_{\bar{f}_2}$
         - $y_i^j \leftarrow T^R_{\bar{f}_2}$
      iii. $\bar{f}_2(y_{i-1}^j, M_i^j) \leftarrow y_1^j$
5. $y_N^j \overset{\$}{\leftarrow} \{0,1\}^n$
6. if $(y_{N-1}^j, M_N^j) \in T^Q_{\bar{f}_3}$
   - $y_N^j \leftarrow T^R_{\bar{f}_3}$
7. $\bar{f}_3(y_{N-1}^j, M_N^j) \leftarrow y_N^j$
8. **ret** y_N^j

Only for G_6**:** On query to $IH(y_0^j, M_1^j \| ... \| M_N^j)$

1. if $IH(y_0^j, M_1^j \| ... \| M_N^j) = \perp$
   (a) $IH(y_0^j, M_1^j \| ... \| M_N^j) \overset{\$}{\leftarrow} \{0,1\}^n$
2. **ret** $IH(y_0^j, M_1^j \| ... \| M_N^j)$

On query to $\bar{f}_1(y_{l-1}^j, M_l^j)$

1. $y_l^j \overset{\$}{\leftarrow} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T^Q_{\bar{f}_1}$
   - $y_l^j \leftarrow T^R_{\bar{f}_1}$
   - **ret** y_l^j
3. $NewPath(y_{l-1}^j, \bar{f}_1, M_l^j) \leftarrow y_l^j$
4. $\bar{f}_1(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
5. if $(y_l^j \in T^R_{\bar{f}_1}) \vee (y_l^j \in T^{Q_y}_{\bar{f}_2}) \vee (y_l^j \in T^R_{\bar{f}_2}) \vee (y_l^j \vee T^{Q_y}_{\bar{f}_3})$
   - $bad_{\bar{f}_1} \leftarrow true$
   - $y_l^j \overset{\$}{\leftarrow} \{0,1\}^n \backslash (T^R_{\bar{f}_1} \bigcup T^{Q_y}_{\bar{f}_2} \bigcup T^R_{\bar{f}_2} \bigcup T^{Q_y}_{\bar{f}_3})$
6. **ret** y_l^j

On query to $\bar{f}_2(y_{l-1}^j, M_l^j)$

1. $y_l^j \overset{\$}{\leftarrow} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T^Q_{\bar{f}_2}$
   - $y_l^j \leftarrow T^R_{\bar{f}_2}$
   - **ret** y_l^j
3. else
   - $(y_0^j, M_r^j) \leftarrow GetPath(y_{l-1}^j)$
   - $NewPath(y_0^j, \bar{f}_2, M_r^j \| M_l^j) \leftarrow y_l^j$
4. if $y_l^j \in (T^R_{\bar{f}_1}) \vee (T^R_{\bar{f}_2}) \vee (T^{Q_y}_{\bar{f}_2}) \vee (T^{Q_y}_{\bar{f}_3})$
   - $bad_{\bar{f}_2} \leftarrow true$
   - $y_l^j \overset{\$}{\leftarrow} \{0,1\}^n \backslash (T^R_{\bar{f}_1} \bigcup T^R_{\bar{f}_2} \bigcup T^{Q_y}_{\bar{f}_2} \bigcup T^{Q_y}_{\bar{f}_3})$
5. **ret** y_l^j

On query to $\bar{f}_3(y_{l-1}^j, M_l^j)$

1. $y_l^j \overset{\$}{\leftarrow} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T^Q_{\bar{f}_3}$
   - $y_l^j \leftarrow T^R_{\bar{f}_3}$
   - **ret** y_l^j
3. else if $(y_0^j, M_r^j) \leftarrow GetPath(y_{l-1}^j)$ then
   - $y_l^j \leftarrow IH(y_0^j, M_r^j \| M_l^j)$
4. $\bar{f}_3(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
5. **ret** y_l^j

$$Pr[\mathcal{A}^{G_5} \Rightarrow (bad \leftarrow true)] \leq Pr_{G_5}^{bad_{\bar{f}_1}} + Pr_{G_5}^{bad_{\bar{f}_2}}$$

Now we bound each of the bad events as follows:

1. $bad_{\bar{f}_1}$: This event is set to true if the current selected random value for a query to $\bar{f}_1$ is collided with a value in $(T^R_{\bar{f}_1} \bigcup T^{Q_y}_{\bar{f}_2} \bigcup T^R_{\bar{f}_2} \bigcup T^{Q_y}_{\bar{f}_3})$. For the ith query to $\bar{f}_1$ we have $i-1$ domain and range points defined for $\bar{f}_1$ and up to $q^{\bar{f}_2}$ (resp. $q^{\bar{f}_3}$) previous queries to $\bar{f}_2$ (resp. $\bar{f}_3$). Thus:

   $$\left|(T^R_{\bar{f}_1} \bigcup T^{Q_y}_{\bar{f}_2} \bigcup T^R_{\bar{f}_2} \bigcup T^{Q_y}_{\bar{f}_3})\right| \leq i - 1 + 2q^{\bar{f}_2} + q^{\bar{f}_3}$$

   The probability that one of the values that set this bad event to true is selected at random from $\{0,1\}^n$ for the ith query to $\bar{f}_1$ is not larger than $(i - 1 + 2q^{\bar{f}_2} + q^{\bar{f}_3})/2^n$. Hence, we can sum up this probability over all the queries to $\bar{f}_1$ and bound the probability of $bad_{\bar{f}_1}$ occurrence, $Pr_{G_5}^{bad_{\bar{f}_1}}$, as follows:

   $$Pr_{G_5}^{bad_{\bar{f}_1}} \leq \sum_{i=1}^{q^{\bar{f}_1}} \frac{(i - 1 + 2q^{\bar{f}_2} + q^{\bar{f}_3})}{2^n}$$

   $$= \frac{1}{2^n}\left(\sum_{i=1}^{q^{\bar{f}_1}} i - \sum_{i=1}^{q^{\bar{f}_1}} 1 + 2q^{\bar{f}_2}\sum_{i=1}^{q^{\bar{f}_1}} 1 + q^{\bar{f}_3}\sum_{i=1}^{q^{\bar{f}_1}} 1\right)$$

   $$\leq \frac{(q^{\bar{f}_1})^2 + 2q^{\bar{f}_1}(2q^{\bar{f}_2} + q^{\bar{f}_3})}{2^{n+1}}$$

   $$\leq \frac{q^{\bar{f}_1}(q^{\bar{f}_1} + 2q^{\bar{f}_2} + q^{\bar{f}_3})}{2^n}$$

Fig. 8 G_7 representation

On query (y_0^j, M^j) to $\bar{H^f}$

1. if $F(y_0^j, M^j) = \perp$
   (a) $F(y_0^j, M^j) \xleftarrow{\$} \{0,1\}^n$
2. **ret** $F(y_0^j, M^j)$

On query to $\bar{f}_1(y_{l-1}^j, M_l^j)$

1. $y_l^j \xleftarrow{\$} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_1}^Q$
   – $y_l^j \leftarrow T_{\bar{f}_1}^R$
   – **ret** y_l^j
3. $NewPath(y_{l-1}^j, \bar{f}_1, M_l^j) \leftarrow y_l^j$
4. $\bar{f}_1(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
5. if $(y_l^j \in T_{\bar{f}_1}^R) \vee (y_l^j \in T_{\bar{f}_2}^{Q_y}) \vee (y_l^j \in T_{\bar{f}_2}^R) \vee (y_l^j \vee T_{\bar{f}_3}^{Q_y})$
   – $bad_{\bar{f}_1} \leftarrow true$
   – $y_l^j \xleftarrow{\$} \{0,1\}^n \backslash (T_{\bar{f}_1}^R \bigcup T_{\bar{f}_2}^{Q_y} \bigcup T_{\bar{f}_2}^R \bigcup T_{\bar{f}_3}^{Q_y})$
6. **ret** y_l^j

On query to $\bar{f}_2(y_{l-1}^j, M_l^j)$

1. $y_l^j \xleftarrow{\$} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_2}^Q$
   – $y_l^j \leftarrow T_{\bar{f}_2}^R$
   – **ret** y_l^j
3. else
   – $(y_0^j, M_r^j) \leftarrow GetPath(y_{l-1}^j)$
   – $NewPath(y_0^j, \bar{f}_2, M_r^j \| M_l^j) \leftarrow y_l^j$
4. if $y_l^j \in (T_{\bar{f}_1}^R) \vee (T_{\bar{f}_2}^R) \vee (T_{\bar{f}_2}^{Q_y}) \vee (T_{\bar{f}_3}^{Q_y})$
   – $bad_{\bar{f}_2} \leftarrow true$
   – $y_l^j \xleftarrow{\$} \{0,1\}^n \backslash (T_{\bar{f}_1}^R \bigcup T_{\bar{f}_2}^R \bigcup T_{\bar{f}_2}^{Q_y} \bigcup T_{\bar{f}_3}^{Q_y})$
5. **ret** y_l^j

On query to $\bar{f}_3(y_{l-1}^j, M_l^j)$

1. $y_l^j \xleftarrow{\$} \{0,1\}^n$
2. if $(y_{l-1}^j, M_l^j) \in T_{\bar{f}_3}^Q$
   – $y_l^j \leftarrow T_{\bar{f}_3}^R$
   – **ret** y_l^j
3. else if $(y_0^j, M_r^j) \leftarrow GetPath(y_{l-1}^j)$ then
   – $y_l^j \leftarrow \bar{H^f}(y_0^j, M_r^j \| M_l^j)$
4. $\bar{f}_3(y_{l-1}^j, M_l^j) \leftarrow y_l^j$
5. **ret** y_l^j

2. $bad_{\bar{f}_2}$: This event is set to true if the current selected random value for a query to $\bar{f}_2$ is collided with a value in $(T_{\bar{f}_1}^R \bigcup T_{\bar{f}_2}^{Q_y} \bigcup T_{\bar{f}_2}^R \bigcup T_{\bar{f}_3}^{Q_y})$. Hence, based on the calculation of $bad_{\bar{f}_1}$, we can bound the probability of $bad_{\bar{f}_2}$ occurrence, $Pr_{G_5}^{bad_{\bar{f}_2}}$, as follows:

$$\begin{aligned} Pr_{G_5}^{bad_{\bar{f}_2}} &\leq \sum_{i=1}^{q^{\bar{f}_2}} \frac{(2(i-1) + q^{\bar{f}_1} + q^{\bar{f}_3})}{2^n} \\ &= \frac{(q^{\bar{f}_2})^2 + q^{\bar{f}_2}(q^{\bar{f}_1} + q^{\bar{f}_3})}{2^n} \\ &= \frac{q^{\bar{f}_2}(q^{\bar{f}_1} + q^{\bar{f}_2} + q^{\bar{f}_3})}{2^n} \end{aligned}$$

Thus:

$$\begin{aligned} &\left| Pr[\mathcal{A}^{G_5} \Rightarrow 1] - Pr[\mathcal{A}^{G_4} \Rightarrow 1] \right| \leq Pr_{G_5}^{bad_{\bar{f}_1}} + Pr_{G_5}^{bad_{\bar{f}_2}} \\ &\leq \frac{q^{\bar{f}_1}(q^{\bar{f}_1} + 2q^{\bar{f}_2} + q^{\bar{f}_3})}{2^n} + \frac{q^{\bar{f}_2}(q^{\bar{f}_1} + q^{\bar{f}_2} + q^{\bar{f}_3})}{2^n} \\ &\leq \frac{(q^{\bar{f}_1} + q^{\bar{f}_2})(q^{\bar{f}_1} + 2q^{\bar{f}_2} + q^{\bar{f}_3})}{2^n} \end{aligned}$$

– **Game 6** (G_6)**:** In game G_6, C_H changes IH such that it simply return a random value for any new query. The implementation of $IH(y, M)$ in G_5 follows the $SFPF$ iteration, while G_6 returns a random value for any new query to $\bar{H^f}$. It is obvious that the returned values for the queries to $\bar{H^f}$ in G_6 and G_5 are determined by the IH- sub function. Both games return random values for any new query (y, M) to $\bar{H^f}$ where M consists of N message blocks M_i for $i = 1, \ldots, N$. G_5 answers such queries by invoking $\bar{f}_1(y_0, M_1)$, $\bar{f}_2(y_{i-1}, M_i)$ for $2 \leq i \leq N-1$, and $\bar{f}_3(y_{N-1}, M_N)$ in the order. Whereas for any new query (y, M) to $\bar{H^f}$, G_6 does not invoke $\bar{f}_1$, $\bar{f}_2$ and $\bar{f}_3$ and selects the answer to such new queries randomly from $\{0,1\}^n$. Hence, in G_6 the cardinality of $T_{\bar{f}_1}^R$, $T_{\bar{f}_2}^{Q_y}$, $T_{\bar{f}_2}^R$, and $T_{\bar{f}_3}^{Q_y}$ would be decreased up to $q^{\bar{H^f}}$, $(\tau-2)q^{\bar{H^f}}$, $(\tau-2)q^{\bar{H^f}}$ and $q^{\bar{H^f}}$ respectively, which reduces the probability of receiving a bad event in G_6 compared to that probability in G_5. Hence, we have:

$$\begin{aligned} &\left| Pr[\mathcal{A}^{G_6} \Rightarrow 1] - Pr[\mathcal{A}^{G_5} \Rightarrow 1] \right| \leq \left| Pr_{G_5}^{bad_{\bar{f}_1}} - Pr_{G_6}^{bad_{\bar{f}_1}} \right| \\ &\quad + \left| Pr_{G_5}^{bad_{\bar{f}_2}} - Pr_{G_6}^{bad_{\bar{f}_2}} \right| \leq Pr_{G_5}^{bad_{\bar{f}_1}} + Pr_{G_5}^{bad_{\bar{f}_2}} \\ &\leq \frac{(q^{\bar{f}_1} + q^{\bar{f}_2})(q^{\bar{f}_1} + 2q^{\bar{f}_2} + q^{\bar{f}_3})}{2^n} \end{aligned}$$

– **Game 7** (G_7)**:** We finish the play with the "ideal" game G_7 that exactly simulates R and S^f. In this game, $\bar{H^f}$ does not send its query to IH any more and respond to any new query randomly. However, it has no affect on the returned values to $\mathcal{A}$. Thus, in G_7, $\mathcal{A}$ does not gain any additional advantage over G_6 and $p_7 = p_6$. In this

game, $\bar{H}^f$ is exactly the same as R, and the controller C_H is precisely equivalent to S^f, our proposed simulator for f_1, f_2 and f_3.

We complete the proof by combining Games 0 to 7. Note that G_0 emulates H^f and $f_1 f_2$ and f_3 and G_7 exactly emulates R and S^f. We conclude that:

$$\begin{aligned} Adv^{indif}_{R,S}(\mathcal{A}) &= \left| Pr[\mathcal{A}^{H^f,(f_1,f_2,f_3)}] - Pr[\mathcal{A}^{R,S^f}] \right| \\ &\leq 2 \times (Pr^{bad_{\bar{f}_1}}_{G_5} + Pr^{bad_{\bar{f}_2}}_{G_5}) \\ &\leq 2 \times \frac{(q^{\bar{f}_1} + q^{\bar{f}_2})(q^{\bar{f}_1} + 2q^{\bar{f}_2} + q^{\bar{f}_3})}{2^n} \end{aligned}$$

With further simplification, this results in

$$Adv^{indif}_{R,S}(\mathcal{A}) = \epsilon \leq \frac{4q^2}{2^n}$$

□

10 Appendix 3: Simulator for the SFPF-N hash function

In this section, we present a simulator for the SFPF-N hash function in Algorithm 11. This simulator emulates f_1 and f_2 such that SFPF-N is indifferentiable from R. For simplicity and without loss generality, this simulator assumes that the entire last block is used for MD strengthening. Its running time $t_S = O(q^2)$, and $\mathcal{A}$'s advantage after q queries is bounded by $\epsilon \leq O(\tau^2 \cdot q^2 \cdot 2^{-n})$.

Algorithm 11: A Simulator for the SFPF-N Hash Function

Input:
- query $QR_i^q = \bar{f}_i(y_{l-1}^j, M_l^j)$ // f_i could be either f_1 or f_2
- table $T_{\bar{f}_i} : T^Q_{\bar{f}_i} \rightarrow T^R_{\bar{f}_i}$ for $i = 1, 2$ of old query relations

$QR_i^r \xleftarrow{\$} \{0,1\}^n$ **if** $QR_i^q \in T^Q_{\bar{f}_i}$ **then** // old query?
  $QR_i^r \leftarrow T^R_{\bar{f}_i}$ // return table entry
else
  if $(\bar{f}_i == \bar{f}_1)$ **then** // new query to $\bar{f}_1$
    - $NewPath(y_{l-1}^j, \bar{f}_1, M_l^j) \leftarrow QR_i^r$ // update the tree
    - if $QR_i^r \in (T^R_{\bar{f}_1}) \vee (T^R_{\bar{f}_2}) \vee (T^{Q_y}_{\bar{f}_2})$:
      - $bad_{\bar{f}_1} \leftarrow true$
      - $QR_i^r \xleftarrow{\$} \{0,1\}^n \backslash (T^R_{\bar{f}_1}) \vee (T^R_{\bar{f}_2}) \vee (T^{Q_y}_{\bar{f}_2})$

  if $(\bar{f}_i == \bar{f}_2) \wedge ((y_0^j, M_r^j) \leftarrow GetPath(y_{l-1}^j)) \wedge \left(\frac{|M_r^j|}{m} == M_l^j\right)$ **then** // new query to $\bar{f}_2$
    $QR_i^r \leftarrow R(y_0^j, M_r^j \| M_l^j)$ // to consult R
  $T^R_{\bar{f}_i} \leftarrow QR_i^r$
return $T^R_{\bar{f}_i}$

11 Appendix 4: Formal description of the Games used in the indifferentiability analysis of the SFPF hash function

In this section, we provide figures that formally describe the games used in the indifferentiability analysis of the SFPF hash function. See Figs. 4, 5, 6, 7, 8.

References

1. Bellare, M., Canetti, R., Krawczyk, H.: Pseudorandom functions revisited: the cascade construction and its concrete security. In: Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, FOCS'96, pp. 514–523. IEEE Computer Society, IEEE Computer Society Press (1996)
2. Bellare, M., Ristenpart, T.: Multi-property-preserving hash domain extension and the EMD transform. In: Proceedings of ASIACRYPT 2006, vol. 4284 of Lecture Notes in Computer Science, pp. 299–314. Springer (2006)
3. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of CCS '93p, pp. 62–73. ACM Press (1993)
4. Chang, D., Lee, S., Nandi, M., Yung, M.: Indifferentiable security analysis of popular hash functions with prefix-free padding. In: Proceedings of ASIACRYPT 2006, vol. 4284 of Lecture Notes in Computer Science, pp. 283–298. Springer (2006)
5. Chang, D., Nandi, M.: Improved Indifferentiability Security Analysis of chopMD Hash Function. In: Proc. FSE 2008, volume 5086 of Lecture Notes in Computer Science, pp. 429–443. Springer (2008)
6. Chang, D., Sung, J., Hong, S., Lee, S.: Indifferentiable security analysis of choppfMD, chopMD, chopMDP, chopWPH, chopNI, chopEMD, chopCS, and chopESh hash domain extensions. Cryptology ePrint archive, report 2008/407 (2008)
7. Coron, J.-S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle–Damgård revisited: how to construct a hash function. In: Proceedings of CRYPTO 2005, vol. 3621 of Lecture Notes in Computer Science, pp. 430–448. Springer (2005)
8. Coron, J.-S., Patarin, J., Seurin, Y.: The random oracle model and the ideal cipher model are equivalent. In: Proceedings of CRYPTO 2008, volume 5157 of Lecture Notes in Computer Science, pp. 1–20. Springer (2008)
9. Damgård, I.B.: A design principle for hash functions. In: Proceedings of CRYPTO 1989, vol. 435 of Lecture Notes in Computer Science, pp. 416–427. Springer (1989)
10. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: a strengthened version of RIPEMD. In: Proceedings of FSE 1996, vol. 1039 of Lecture Notes in Computer Science, pp. 71–82. Springer (1996)
11. Gong, Z., Lai, X., Chen, K.: A synthetic indifferentiability analysis of some block-cipher-based hash functions. Des. Codes Cryptogr. **48**(3), 293–305 (2008)
12. Hirose, S., Park, J.H., Yun, A.: A simple variant of the Merkle–Damgård scheme with a permutation. In: Proceedings of ASIACRYPT 2007, vol. 4833 of Lecture Notes in Computer Science, pp. 113–129. Springer (2007)
13. Joux, A.: Multicollisions in iterated hash functions. Application to cascaded constructions. In: Proceedings of CRYPTO 2004, vol. 3152 of Lecture Notes in Computer Science, pp. 306–316. Springer (2004)
14. Kelsey, J., Kohno, T.: Herding hash functions and the Nostradamus attack. In: Proceedings of EUROCRYPT 2006, vol. 4004 of Lecture Notes in Computer Science, pp. 183–200. Springer (2006)

15. Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2^n work. In: Proceedings of EUROCRYPT 2005, vol. 3494 of Lecture Notes in Computer Science, pp. 474–490. Springer (2005)
16. Lai, X., Massey, J.L.: Hash functions based on block ciphers. In: Proceedings of EUROCRYPT 1992, vol. 658 of Lecture Notes in Computer Science, pp. 53–66. Springer (1992)
17. Lucks, S.: A failure-friendly design principle for hash functions. In: Proceedings of ASIACRYPT 2005, vol. 3788 of Lecture Notes in Computer Science, pp. 474–494. Springer (2005)
18. Maurer, U.M., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Proceedings of TCC '04, vol. 2951 of Lecture Notes in Computer Science, pp. 21–39. Springer (2004)
19. Merkle, R.C.: One way hash functions and DES. In: Proceedings of CRYPTO 1989, vol. 435 of Lecture Notes in Computer Science, pp. 428–446. Springer (1989)
20. National Institute of Standards and Technology.: FIPS PUB 180–2-Secure Hash Standard, Aug 2002
21. Preneel, B.: Analysis and design of cryptographic hash functions. Thesis (Ph.D.), Katholieke Universiteit Leuven, Leuven, Belgium, Jan 1993